Looking under the Hood of Stochastic Machine Learning Algorithms for Parts of Speech Tagging

Jana Diesner Kathleen M. Carley

July 2008
CMU-ISR-07-131R

Institute for Software Research
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Center for the Computational Analysis of Social and Organizational Systems
CASOS technical report.
This report supersedes the previous report, CMU-ISR-08-131.

This work was supported in part by the Army Research Lab as part of the CTA in Decision Making DAAD19-01-2-0009, the Army Research Institute W91WAW07C0063, and the National Science Foundation IGERT 9972762 in CASOS. Additional support was provided by CASOS and ISR at Carnegie Mellon University. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Lab, the Army Research Institute, the National Science Foundation, or the U.S. government.. We are grateful to Alex Rudnicky, CMU, for providing the data to us, to Yifen Huang, CMU, for discussing the project with us, and to Carolyn Rose, CMU, and Jamie Olson, CMU, for their comments on this paper.

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **JUL 2008**	2. REPORT TYPE	3. DATES COVERED **00-00-2008 to 00-00-2008**

4. TITLE AND SUBTITLE **Looking under the Hood of Stochastic Machine Learning Algorithms for Parts of Speech Tagging**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Carnegie Mellon University,School of Computer Science,Institute for Software Research,Pittsburgh,PA,15213**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
A variety of Natural Language Processing and Information Extraction tasks, such as question answering and named entity recognition, can benefit from precise knowledge about a words? syntactic category or Part of Speech (POS) (Church, 1988; Rabiner, 1989; Stolz, Tannenbaum, & Carstensen, 1965). POS taggers are widely used to assign a single best POS to every word in text data, with stochastic approaches achieving accuracy rates of up to 96% to 97% (Jurafsky & Martin, 2000). When building a POS tagger, human beings needs to make a set of choices about design decisions, some of which significantly impact the accuracy and other performance aspects of the resulting engine. However, documentations of POS taggers often leave these choices and decisions implicit. In this paper we provide an overview on some of these decisions and empirically determine their impact on POS tagging accuracy. The gained insights can be a valuable contribution for people who want to design, implement, modify, fine-tune, integrate, or responsibly use a POS tagger. We considered the results presented herein in building and integrating a POS tagger into AutoMap, a tool that facilitates relation extraction from texts, as a stand-alone feature as well as an auxiliary feature for other tasks.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**35**	

Keywords: Part of Speech Tagging, Hidden Markov Models, Viterbi Algorithm, AutoMap

Abstract

A variety of Natural Language Processing and Information Extraction tasks, such as question answering and named entity recognition, can benefit from precise knowledge about a words' syntactic category or Part of Speech (POS) (Church, 1988; Rabiner, 1989; Stolz, Tannenbaum, & Carstensen, 1965). POS taggers are widely used to assign a single best POS to every word in text data, with stochastic approaches achieving accuracy rates of up to 96% to 97% (Jurafsky & Martin, 2000). When building a POS tagger, human beings needs to make a set of choices about design decisions, some of which significantly impact the accuracy and other performance aspects of the resulting engine. However, documentations of POS taggers often leave these choices and decisions implicit. In this paper we provide an overview on some of these decisions and empirically determine their impact on POS tagging accuracy. The gained insights can be a valuable contribution for people who want to design, implement, modify, fine-tune, integrate, or responsibly use a POS tagger. We considered the results presented herein in building and integrating a POS tagger into AutoMap, a tool that facilitates relation extraction from texts, as a stand-alone feature as well as an auxiliary feature for other tasks.

Table of Contents

1. Introduction 1
2. Method 2
3. Data 5
4. Experiment 6
   4.1 Disassembling Viterbi 6
   4.2 Handling Noise 8
   4.3 Smoothing and Handling of Unknown Data Points 9
   4.4 Aggregating Hidden States 11
5. Results 12
   5.1 Disassembling Viterbi 12
   5.2 Handling Noise 13
   5.3 Handling Unknowns 14
   5.4 Aggregation of Hidden States 19
6. Integration of Parts of Speech Tagging into AutoMap 21
7. Limitations and Conclusions 25

1. Introduction

Part of Speech Tagging (POST) assigns a single best part of speech (POS), such as *noun*, *preposition* or *personal pronoun*, to every word in a text or text collection. What is the knowledge about words' lexical category useful for? First, a large variety of Natural Language Processing (NLP) and Information Extraction (IE) tasks can benefit from accurate knowledge about words' lexical categories, such as:

- Stemming (conversion of terms into their morphemes) (Krovetz, 1995; Porter, 1980)
- Named Entity Extraction (identification of relevant types of information that are referred to by a name, such as people, organizations, and locations) (Bikel, Schwartz, & Weischedel, 1999)
- Anaphora resolution (conversion of personal pronouns into the actual entities that those pronouns refer to) (Lappin & Leass, 1994)
- Creation of positive (thesaurus) and negative (delete list) filters (Diesner & Carley, 2004)
- Ontological text coding (classification of relevant types of information according to an ontology or taxonomy) (Diesner & Carley, 2008)

Second, POS are often used as one feature for machine learning tasks that involve text data (Arguello & Rose, 2006; Bikel, et al., 1999).

What is the challenge in POST? While many words can be unambiguously associated with one tag, e.g. *computer* with noun, other words match multiple tags, depending on the context that they appear in. *Wind*, for example, can be a noun in the context of weather, and can be a verb that refers to coiling something. DeRose (DeRose, 1988) for example reports that in the Brown corpus, which is part of the data set that we use in this study, over 40% of the words are syntactically ambiguous. This example illustrates the fact that ambiguity resolution is the key challenge in POST.

The goal with this report is two-fold: The first one is based on our observation that while many detailed descriptions of POST algorithms exists, several design decisions that need to be made when implementing these algorithms are left implicit in these descriptions. As we demonstrate herein, different choices for these decisions can significantly impact the performance of the tagger. For this project, we operationalize performance as POST accuracy. Therefore, the first goal with this report is to describe a set of design decisions and possible choices in detail, and determining the isolated impact of these choices on POST accuracy. Who cares about such information? We envision the knowledge about the sensitivity of the resulting engine and its parts to be valuable information for people who build taggers, who integrate existing taggers into a system, or who use off-the-shelve taggers.

The second goal with this report is a practical, need-driven one: at the Center for

Computational Analysis of Social and Organizational Systems (CASOS) at Carnegie Mellon University (CMU) we have developed AutoMap, a tool that facilitates the extraction of relational data from texts (Diesner & Carley, 2004; McConville, Diesner, & Carley, 2008). A variety of NLP and IE routines, such as those listed above, are part of that process. Therefore, a highly accurate POS Tagger is a crucial auxiliary tool for multiple routines in AutoMap. Furthermore, we envision high-quality tagging to serve as a helpful stand-alone feature in AutoMap. In order to build a tagger whose design is transparent to all parties involved and is based on informed design decisions we needed detailed knowledge about the subtleties of POST beyond general algorithmic descriptions.

This report is structured as follows: In section two we select and describe a POST algorithm for this project. Section three describes the dataset based on which we trained and tested various POS taggers. Section four explains four design decisions that need to be made when implementing a POS Tagger, and derives hypotheses on the impact of different choices for these decisions on the resulting POST accuracy. Section five tests our hypotheses in an empirical fashion. Section six shows how we used the gained insights in order to build a POST tagger and integrated it into AutoMap, and highlights various uses of POST for end-users. The paper concludes in a description of applicable limitations.

2. Method

What computational approach should be used for building a POS tagger? Taggers can be divided into rule-based, stochastic and transformation-based systems (Manning & Schütze, 1999). For this project we focus on stochastic taggers, which exploit the power of probabilities and machine learning techniques in order to disambiguate and tag sequences of words (Bikel, et al., 1999; Stolz, et al., 1965). One highly successfully and widely applied approach to statistical modeling of natural language data are Hidden Markov Models (HMM) (Baum, 1972), which are explained in more detail in this section. In the domain of speech recognition for instance, HMM have become the favored model (Rabiner, 1989). HMM are also used for POST, where the most accurate systems achieve errors rates of less than four percent (Jurafsky & Martin, 2000). Most of the existing HMM-based POS taggers are trained with labeled data (e.g. (DeRose, 1988; Weischedel, Meter, Schwartz, Ramshaw, & Palmucci, 1993)), while a small number of taggers use unlabeled data in order to train a model based on expectation maximization (EM) (e.g.(Kupiec, 1992). Given the performance rates that others have achieved with HMM-based stochastic POS taggers we decided to use this approach for building a POS tagger for AutoMap.

HMM are a probabilistic function of Markov Models (MM). In this section we first briefly describe MM, followed by a short explanation of HMM (for details on MM and HMM see (Baum, 1972; Church, 1988; DeRose, 1988; Rabiner, 1989; Stolz, et al., 1965)). Markov Models (MM) model the probabilities of non-independent events in a linear sequence. Applying this idea to natural language allows us to model language as a dynamic system in which words and their underlying features are not isolated events, but do impact each other.

MM are based on two assumptions: First, MM assume a limited horizon into the past. This means that given a current element in a sequence, future elements are conditionally independent of past elements. In other words, elements depend only on themselves and a few predecessors. The number of predecessors considered is called the order of the HMM. If one decides to look at only the most recent data point (word) from the past, then a first-order MM is applied. Second, MM make the time invariance assumption, meaning that probabilities are stationary (invariant over time). This assumption can be related to the desire for generalizable models that are trained on a specific data set and are later applied to new and unseen data. The time invariance assumption is a theoretical one only. In reality, language is a dynamic system, in that rules (syntax) and elements (vocabulary) emerge and vanish over time, and across places and people.

Relating these two assumptions to POST enables us to exploit and computationally combine every word's probability as well as its local context as given by a word's predecessor(s). HMM, a probabilistic function of MM, brings these two pieces of information together by computing the probability of tag sequence $P(tag_{1\text{-}end})$ that maximizes the likelihood of the product of word probability $P(word_i|tag_j)$ and tag sequence probability $P(tag_j|$ *previous n* $tags_{j\text{-}N})$. Applying HMM to POST means aiming to find the most likely sequence of POS in a given sequence, typically a sentence, for all sequence (sentences) in a text or corpus (Baum, 1972; DeRose, 1988; Stolz, et al., 1965)

In practical POST applications, the true sequence of POS that underlies an observed sequence, e.g. a sentence, is unknown, thus forming the hidden states. A POS tagger aims to find the sequence of hidden states that most likely has generated the observed sequence. This task is referred to as *decoding*, which means that given a set of observations x (words in a sentence) and a model μ (the result of supervised learning) we want to reveal the underlying Markov chain of tags that is linked to the observed states. Model μ consists of three parameters:

1. Initial state probabilities π. This is a vector that quantifies the probability of the tag of the first hidden state in a sentence. Why is that needed? When POST is performed on the sentence level (the classical approach), the first word in the sequence has no predecessor. In order to decode this token, it is typically assumed that the most frequently observed tag for this token across the data set is the most likely tag for this token.
2. State transition probabilities a_{ij}, stored in a transition matrix, quantify the likelihood of observing a certain hidden state given the previous hidden state.
3. State emission probabilities b_{ij}, stored in a confusion or emission matrix, specify the probabilities of observing a particular state (word) while the HMM is in a certain hideen state (tag).

When training a POS tagger in a supervised fashion, the parameters of model μ are computed from the training data. Therefore, the process of estimating parameters during model training is a visible Markov process, because the surface pattern (word sequence) and underlying states (POS sequence) can be fully observed. In contrast to that, applying the trained model to tag new and unseen data truly represents a hidden MM, because the tag sequence is hidden underneath the surface pattern and will be revealed using previously gathered empiric evidence (model μ).

The vast majority of HMM practical applications deploy first-order models. This seems counterintuitive if one believes that higher-order MM could lead to more accurate predictions than lower order models, because state sequences might depend not only on one (first-order HMM), but multiple predecessors (e.g. in *Department of Labor*). A time horizon of greater than one, however, results in less and sparser training data due to the lack of local histories for the beginning of sequences (Manning & Schütze, 1999). This translates into a serious disadvantage if sentences in the training data are rather short, or if comas are used as delimiters instead of sentence marks. Because a shift from a first-order HMM to a second-order HMM reduces the amount of training data available and therefore also the numerical stability of the constructed model we decided to work with a first-order HMM. While the limited horizon assumption enables us to account for the fact that the words in a sentence may depend on each other, especially in the case of meaningful bigrams such as *human rights*, it excludes the consideration of long-range dependencies (Diesner & Carley, 2008). Long-range dependencies are not meaningful N-grams whose elements co-occur next to each other, but elements that interact without being collocated, such as personal pronouns that refer back to a social entity mentioned earlier in the text. This limitation has been shown to be a serious weakness if relevant data points are sparsely scattered across the data. Since in POST training data every word has a tag, this limitation does not apply to POST.

Different algorithms for implementing a HMM exist. A widely used one in the NLP domain is the Viterbi algorithm (Viterbi in the following) (Viterbi, 1967). The solution that a POS tagger will suggest is contained in the search space of the applied algorithm or technique. A search space describes and confines the room of possible solutions. For Viterbi, the search space can be represented as a trellis. A trellis is a field composed of a chain of tokens (chain length depends on the number of tokens per sequence) and a related matrix of all hidden states that were empirically observed during model construction by the probabilistic connections (transitions) between the hidden states. The chain of observed states and the matrix of hidden state transitions are probabilistically connected via the empirically observed emission probabilities for a word by the full set of tags. Viterbi's basic idea and main advantage are the reduction of the complexity of examining every full path through a trellis (all possible combinations of tag transitions and word emissions in a sequence) by recursively finding partial probabilities δ for the most likely path from one state to the next throughout each sequence. Viterbi requires three steps for searching and identifying one complete and the most probable route through the trellis:

Viterbi Algorithm, Goal:

Finding the sequence of hidden states that generates the maximum partial probability $\delta_j(t)$ of possible state combination while moving through the trellis:

$$\delta_j(t) = \max_X P(X, O, X_t = j \mid \mu)$$

where

j… index of potential state

t…index in the sequence of observations

$X = X_1 \ldots X_{t-1}$ …sequence of (hidden) states

$O = O_1 \ldots O_{t-1}$ …sequence of observations

The following steps will be executed in order to achieve the goal:

1. Initialization $\delta_j(1) = \pi_j, 1 \le j \le N$
2. Induction $\delta_j(t+1) = \max_{1 \le i \le N} \delta_j(t)\, a_{ij}\, b_{ijo_t}, 1 \le j \le N$

   Store backtrace $\psi_j(t+1) = \arg\max_{1 \le i \le N} \delta_j\, a_{ij}\, b_{ijo_t}, 1 \le j \le N$

   where $\psi_j(t)$ = storage of node of incoming arc to most probable path
3. Termination and path (most likely tag sequence) readout (by backtracking)

$$\hat{X}_{T+1} = \arg\max_{1 \le i \le N} \delta_i(T+1)$$

$$\hat{X}_t = \psi_{\hat{X}_{T+1}}(t+1)$$

$$P(\hat{X}) = \max_{1 \le i \le N} \delta_i(T+1)$$

In summary, the supervised, sequential, stochastic machine learning technique described herein constructs a model *μ* that for each sequence of (*x,y*), where *x* are the words in a sentence and *y* the corresponding POS tags, predicts a POS sequence *y* = *μ(x)* for any sequence of *x*, including new and unseen text data. Note that machine learners are systems that improve their performance (here, POS tagging accuracy) with experience (here, observing token-tag tuples along sentences). Since HMM estimate a joint probability (the one of words and tags) they are a member of the family of generative models (Dietterich, 2002). The tag sequence that results from applying model *μ* to new data may not necessarily be the correct one, but it will be the most likely one given the model and the data. It is for this reason that informed design and implementation of a tagger and careful preparation of the learning and validation data are key to success.

3. Data

The data set used for training and validation in this project is the tagged version of the Penn Treebank 3 (PTB) corpus (Mitchell, Santorini, & Marcinkiewicz, 1993). The PTB collection contains 2,499 texts from differenet sources such as over three years of news coverage from the Wall Street Journal (1989-1992) and a tagged version of the Brown corpus (1961). Every word in the corpus is annotated with at least one out of 36 possible tags (see the Appendix for a list of tags and their meaning). The PTB data is stored in 500 data files, which are organized in 15 folders.

In cases where the human coders who annotated the PTB texts with POS were uncertain about the best POS for a word, e.g. when a word was syntactically ambiguous, multiple tags were assigned in a non-standardized order (Klein & Manning, 2002). For example, *England-born/NNP/VBN* means that England-born might be a singular proper noun as well as past-participle verb. There is a total of 121 such cases of tag indeterminacy in PTB. We performed several qualitative checks (human reasoning about the best out of the offered tags) on randomly drawn instances of this issue from PTB, which convinced us of the random order of multiple tags per word.

4. Experiment

We conducted a series of experiments in order to identify the impact of several independent variables, which we explain in detail this section, on the dependent variable of interest: the accuracy of tagging new data by using the constructed model. What can the outcome of this exercise be useful for? First, we envision creators and users of HMM implementations to use this knowledge in order to build or responsibly apply such systems. Second, we need such detailed information in order to construct the best POST model for AutoMap (for machine learning problems, the best model is typically the most concise one that generalizes with highest accuracy to new data).

4.1 Disassembling Viterbi

Section 2 described the different computational steps that are involved in the Viterbi algorithm. How much accuracy gain can be attributed to each of these steps? In order to answer this question we isolated each step and ran experiments in order to quantify the partial accuracy gain that the following steps accounts for:

1. Probabilities of words in isolation
2. Emission and Transition Probabilities
3. Partial probabilities δ and back pointers ψ
4. Backtracing

Step 1 enables us to isolate and measure the accuracy achieved by using emission probabilities only. This procedure disregards the impact the POS of the proceeding word on the subsequent word's POS, thus not making use of a words' historical context (a "zero-order HMM"). As a result, the tag that has been observed most frequently for the word under consideration during training will be selected. This step resembles the initialization stage as well as the computation of the initial state probabilities as described in section 2. In HMM and Viterbi, probabilities of words in isolation are used for tagging the first word in every sentence as well as for one-word sentences. We refer to this approach as the Unigram Model (UM in the following), and use UM as our baseline performance measure.

Step 2 represents a regular HMM (HMM in the following). That is the product of emission (of a word by a tag) probabilities and transition (from POS to POS) probabilities as computed during the induction stage of Viterbi. The difference in accuracy rates between step 1 and 2 allows us to isolate and quantify the impact of transition probabilities on tagging accuracy. HMM perform local search. This means that the model decides on the most likely tag for each token (by choosing the POS that maximized the product of the possible transition and emission probabilities between the current and preceding words and their POS) prior to moving on to the next word.

Step 3 is the heart of Viterbi. It combines partial probabilities as computed in step 2 with a forward search for the best (a complete and the most probable) path through the trellis. At each step while moving through the sequence for which the hidden states need to be determined the algorithm computes partial probabilities. These partial probabilities are the product of the emission probability of the potential state, the highest transition probability from the previous possible states, and the partial probability of the previous state that generated the highest transition probability. Hence this algorithm considers the emissions, transitions and the globally optimal sequences of hidden states that are determined while moving through each step in the trellis. In the following we refer to this step as VitF (Viterbi Forward). The difference in accuracy between steps 2 and 3 represents the difference between global search and local forward search.

Step 4 not only computes all possible forward paths through a trellis (as done in step 3) from start to end, but after completing the forward search, it also a) determines the final partial probability of the last state, which represents the optimal solution from global forward search, and b) then backtraces the most probable path through the trellis from the last to the first token. In the following we refer to this step as VitB (Viterbi with backtracing). The difference between step 3 and 4 is the difference between global forward search and global forward/ backward search

In summary, the difference between steps 1 and 2 versus steps 3 and 4 represents the difference between a globally versus locally maximized solution. An actual implementation of the Viterbi algorithm requires all four steps. Each of these steps and in the order as outlined here includes the previous step(s), thereby adding to Viterbi's time and space complexity with every step. This increase in computational complexity is because each step, in the presented order, increases the amount of information or empiric evidence that is comprised in the process of making a decision about the best tag sequence. Based on the information provided in this section we derive the following hypothesis:

Hypothesis 1: POST accuracy increases from step to step, so that:

- *accuracy with HMM is higher than with UM*
- *accuracy with VitF is higher than with HMM*
- *accuracy with VitB is higher than with VitF.*

4.2 Handling Noise

Typically, text data includes various types of noise in varying quantity. What precisely qualifies as noise and how much of it will be normalized or eliminated depends on the goal, resources, and researcher. For this project, tagged tokens are not considered as noise if and only if they are composed of an arbitrarily long sequence of any of the following:

- Characters from a or A to z or Z (regular words)
- Numbers from 0 to 9 (numbers)
- Sentence markers (digits and end of sentences)
- Ampersands (used e.g. in corporation names such as *John Wiley & Sons*)
- Dollar symbols (mainly used to denote monetary values)
- Hyphens (often used to denote genitive markers)
- Dashes (often used in compound words such as *long-term*)

All tagged tokens that are or comprise any symbol not listed above are considered as noise herein. The set of noise terms for this project contains for example tokens whose tag resemble the token (e.g. *:/:*), or most (99.84%) tokens that are tagged as symbol (SYM). Commas are part of the SYM set. Only 0.01% of the tokens tagged as list markers (LS) qualified as noise, while most list markers are actual words or numbers.

Figure 1: Excerpt from PTB Data

Publication/NN
and/CC
distribution/NN
:/:
Volume/NN 1/CD
(/((/(
A[fj]/SYM
)/) of/IN
the/DT seventh/JJ edition/NN

Table 1: Impact of Noise Definition on Transitions

Transitions before symbol removal	*Transitions after symbol removal*
NN - CC	NN - CC
CC - NN	CC – NN
NN - :	**NN – NN**
: - NN	NN – CD
NN – CD	**CD - IN**
CD – (	IN – DT
(- (	DT – JJ
(- SYM	JJ - NN
SYM – IN	
IN – DT	
DT – JJ	
JJ - NN	

For other projects, the tokens and tags that we consider as noise terms might be valuable signals. For data that is stored as coma separated values, for instance, commas would serve as the sequence delimiter. Figure 1 shows an excerpt from a POS-tagged PTB data file in that we printed the tagged elements that we consider as noise in red and bold font. Any word-tag tuple in which one or both elements qualify as noise can be removed prior to learning and model evaluation or not. Table 1 shows the transition probabilities for the example given in Figure 1 with and without performing symbol removal. The transitions that both versions differ in are printed in bold and red font. This example shows that when noise is not removed, more and a higher variety of tag transitions will be learned.

Why could determining the impact of noise removing on POST accuracy matter? For practical POST applications, people are typically not interested in predicting tags for symbols, but only for what is typically considered as content. From a computational as well as practical standpoint, decoding noise requires computational resources, which one might not want to spend. Moreover, including noise into learning and evaluation might dilute the numerical stability of emission and transition probabilities of non-noisy tags, thus decreasing accuracy. Based on the information provided in this section we derive the following hypothesis:

Hypothesis 2: Data cleaning prior to learning and evaluation causes an increase in POST accuracy over learning and evaluating with noisy data for all for algorithms.

4.3 Smoothing and Handling of Unknown Data Points

Any HMM implementation requires cautious handling of small numbers and zero probabilities at various points: first, multiplying and propagating partial probabilities in the induction stage can lead to number underflows. Since UM and HMM disregard partial probabilities, this issue only applies to Viterbi. This problem can be avoided by using the natural log of transition and emission probabilities, and converting the respective multiplications into summations.

Second, words and state sequences that have not been observed in the training data, but do occur in the evaluation data, will cause:

- Zero probability in the induction step of Viterbi. As a result, an entire vertical column in the trellis (all δ for step i) would have zero probabilities, so that the propagation of any path would break.
- Accuracy loss for UM, HMM, VitF, and VitB during model evaluation. This is because tokens that did not occur in the training data but are observed in the evaluation or any other new data will have zero probability of being emitted by any tag, as well as a zero probability of being involved in any tag transition. Typically, the *unknown* tag is initially assigned to these words. Practically, *unknown* never matches the best tag for a word, and therefore increases the tagging error rate. Depending on the algorithm used, *unknowns* account for up to 28% of all tokens when a model trained on one portion of PTB data and is applied to another portion of PTB (detail on

that in section 4.4). Accuracy loss due to not handling unknowns cannot be solved by increasing the amount of training data used, but even models trained on humongous training sets are likely to encounter new words when being applied to unseen data. The issue represents the downside of the time-invariance assumption made for MM: language is a continuously changing system with words emerging and vanishing across time and places, e.g. in the cases of new names of people, places, or products.

We empirically test the impact of handling unknowns on POST accuracy. The following unknown handling strategy is used: Zero probabilities for emissions are prevented by adding tokens newly encountered during evaluation to the emission matrix, tagging them as ”UNKNOWN”, and assigning a minimum probability to them. This intervention prevents the multiplication by zero in the development of the trellis. Zero probabilities for transitions that involve the UNKNOWN tag (*P(t|t=UNKNOWN, P(t=UNKNOWN| t)))* and that have not been observed a priori are caught by assigning a minimum probability to them as well. Initially, we chose a minimum probability that equaled the smallest empirically observed probability in the learning data set. This solution resembles the Adding One strategy (Church, 1988), which in addition to linear interpolation is a frequently applied smoothing technique in tagging (Kupiec, 1992). Later on we realized that in some cases our minimum probability equaled empirically observed probabilities. In cases of ties between any tag and the unknown tag, our engine makes a random choice, which can give an empirically observed small probability (EP) the same weight as the artificially assigned minimum probability (AP). In order to weight EPs higher than APs we decided to first find the smallest EP in the data, dividing it by 100 (we ran multiple tests in order to find an appropriate value), and using the resulting value as the AP. We found that for handling emission probabilities involving unknowns this strategy leads to major, positive changes in accuracy rates, especially for VitF and VitB. For taking care of transitions that comprise the unknown tag, this strategy does not lead to significant changes in POST accurate rates, but it does suppress the detection of unknowns to a degree where they become unlikely to ever be selected. However, in some cases we want to maintain the unknown tag in order to be able to send it to a post-processor, which we explain more detail in section 5.3. It is for this reason that we choose to weight EP deterministically higher than APs only for emissions, but not for transitions.

After zero probabilities for emission and transition have been converted to minimum probabilities lower than EPs, words tagged as *UNKNOWN* are passed to a post-processor, which applies a set of rules in order to tag unknowns as an actual POS. The best-performing unknown-word resolution techniques in tagging use information about the word’s spelling (DeRose, 1988; Viterbi, 1967). We built upon this idea. The construction of the post-processor is described in section 5.3.. Based on the assumption that an actual tag is more likely to be the best tag for a word than the unknown tag, we derive the following hypothesis:

Hypothesis 3: Post-processing of unknown words causes an increase in POST accuracy for all four algorithms.

4.4 Aggregating Hidden States

PTB uses a set of 48 unique tags. 36 of them are regular POS. The other 12 are symbols (#,$,.,,,:,(,),",',",',"). The Appendix lists the regular POS along with the total frequency of their occurrence in PTB. Section 4.2. explained how we handle the symbols. For many real-world applications, the 36 tag classes are too detailed. When analyzing newspaper articles for instance, people often are interested in identifying text terms that refer to the *who*, *what*, *where*, *when*, *why* and *how* of what a report. In that case, the set of singular and plural proper nouns might be a useful starting point for identifying instances of *who* (one or multiple people) and *where* (locations), the union of verbs might help to retrieve the set of words that indicate an action (*what*), and several categories that represent non-content bearing words with respect to the task at hand might be excluded from further consideration. PTB divides for instance verbs into six subgroups (base form verbs, present participle or gerund verbs, present tense not 3rd person singular verbs, present tense 3rd person singular verbs, past participle verbs, past tense verbs), which for some applications we might want to aggregated into one *verb* group. Also, for certain purposes, the union of all prepositions, conjunctions, determiners, possessive pronouns, particles, adverbs, and interjections could be collected into one group that represents irrelevant terms. For this project, we aggregated the regular POS from the PTB tag set into twelve categories as shown in Table 2.

Table 2: Aggregation of PTB Categories

Aggregated Tag	Meaning	Number of Categories in PTB	Instances in PTB
IRR	Irrelevant term	16	409,103
NOUN	Noun	2	217,309
VERB	Verb	6	166,259
ADJ	Adjective	3	81,243
AGENTLOC	Agent	1	62,020
ANA	Anaphora	1	47,303
SYM	Noise	8	36,232
NUM	Number	1	15,178
MODAL	Modal verb	1	14,115
POS	Genetive marker	1	5,247
ORG	Organization	1	1,958
FW	Foreign Word	1	803

The consolidated set comprises personal singular nouns (AGENTLOC), personal plural nouns (ORG), verbs (VERB), modal verbs (MODAL), nouns (NOUN), adjectives (ADJ), personal pronouns (ANA), genitive markers (POS), non-content bearing words (IRR), symbols (NOISE), numbers (NUM), and foreign words (FW). Seven of the aggregated categories map to only one PTB category, while the other categories are represented by up to 16 different PTB categories. The rows in Table 2 are sorted by decreasing frequency of the cumulative occurrence of each category in PTB (last column in Table 2) in order to illustrate the fact that the number of words per tag category varies widely (for details see Appendix).

Our aggregation is one possible solution. For other text sets, domains, or projects, other consolidations might be more appropriate. Based on the assumption that accuracy increases as the pool of choices from which the classifier needs to pick one best POS decreases, we derive the following hypothesis:

Hypothesis 4: Aggregation of POST categories causes an increase in POST accuracy for all four algorithms.

5. Results

The impact of each variable or routine described in section 4 on POST accuracy was tested empirically by performing ten-fold cross validations per variable and averaging the results. In order to enable ten-fold cross validations we randomly split the corpus (500 files, about one million words) into ten folds of equal size (50 files per fold). For each run within a set of ten runs, nine folds are used for training and generating model μ. From the one left-out fold all tags are removed, and μ is applied to this fold in order to tag the data. The assigned tags are then compared to the original labeling of this fold, and every deviation from an original tag is recorded as an error. This procedure is repeated nine more times such that each fold is used once for evaluation and nine times for training, but is never used for training and evaluation at the same time. The results reported in this section were computed by averaging the error rates of ten consecutive runs.

Since the ten folds remain the same across all tests, we performed two-sided paired t-tests in order to determine the statistical significance of the measured difference between any two variables (using a confidence interval of 95%). From an experimental design perspective, the variables that we tested can be considered as independent ones, and their impact on the dependent variable can be tested in isolation. In practical applications of the designed system, these variables interact, and these interdependencies are desired.

Typically, taggers are evaluated with the Gold Standard test, and by comparing the results to a Unigram Baseline test or another benchmark. (Jurafsky & Martin, 2000). The Gold Standard measures performance by identifying the portion of tags that the tagger and a human-labeled validation set agree upon. We use this test for model evaluation. We also use the Unigram Baseline test, which is the same as the performance of the UM model. This model was described in section 2. The highest published accuracy rates for POS taggers that were built by using the PTB are 96% to 97% (Jurafsky & Martin, 2000).

5.1 Disassembling Viterbi

How much partial accuracy can be attributed to the different steps involved in Viterbi? Our findings as shown in Table 3 and 4 indicate that on average, the baseline model (UM) tags 86.83% of the words in the evaluation set correctly. Upgrading from UM to HMM leads to a significant accuracy increase of 5.15% (all significance tests in this paper are based on two-sided, paired t-tests with a 95% confidence interval). This means that considering transitions among hidden states improves predictive power substantially. Further enhancing the implementation to VitF and thereby switching from a locally to a globally maximized solution, results in a much smaller accuracy gain 0.29%. VitB, which out of the algorithms tested exploits the most empiric evidence, achieves another significant 1.02% increase in accuracy; confirming that backtracing does improve Viterbi. The standard deviations (0.36% at the most), which decrease by algorithm, suggest that the results are fairly robust across different portions of the data set.

Table 3: Accuracy per Algorithm

	UM	HMM	VitF	VitB
Average	**86.83%**	**91.98%**	**92.27%**	**93.29%**
Min	86.41%	91.68%	91.95%	93.02%
Max	87.46%	92.38%	92.67%	93.68%
Std Dev	0.36%	0.24%	0.23%	0.23%

Table 4: Difference between algorithms

From	To	Difference	Significance
UM	**HMM**	5.15%	0.00**
HMM	**VitF**	0.29%	0.00**
VitF	**VitB**	1.02%	0.00**

Overall, our empiric results confirm our first hypothesis, which assumed that switching to an algorithm that exploits more evidence than the previous one (UM to HMM, HMM to VitF, VitF to VitB) leads to increased accuracy rates. Also, our findings confirm the previously made observation that the baseline algorithm (UM), which only considers emission probabilities, is a powerful prediction method (Atwell, 1987). Furthermore, we can confirm our assumption that global search outperform local search. However, the difference between HMM and VitF is fairly small (0.29%), and smaller than all other differences between upgrades in algorithms. One possible explanation for this observation is the following chain of thought: HMM weight transition and emission probabilities about equally strong, while both versions of Viterbi weight transitions higher than emissions. VitF enables very small and occasionally meaningless transition probabilities – an effect that VitB partially corrects. The fact that VitF outperforms HMM only slightly suggests that once transitions between hidden states are considered, one needs to go the extra mile of searching a directed web of probabilistic connections among underlying states back *and* forth in order to achieve a substantial gain from global search over local search. Searching through the space of possible solutions not only forward, but forward and backward, has a greater (in our case more than three times greater) impact than considering connections amongst underlying patterns at all.

5.2 Handling Noise

Is it worthwhile cleaning the data from symbols that do not need to be predicted for practical POS applications? For generating the results shown on the previous page we did not remove any symbols. These numbers will now serve as our control case. Disregarding any tokens that contains any element that is not a letter, number, ampersand, dollar symbol, hyphen, or dash for neither learning nor evaluating leads to significant decreases in accuracy for all algorithms, ranging from about a half to more than one percent (Table 5).

Table 5: Impact of Handling Noise on Accuracy

Dataset	Measure	UM	HMM	VitF	VitB
Clean	**Average**	**85.72%**	**91.43%**	**91.62%**	**92.61%**
	Min	85.33%	91.05%	91.29%	92.19%
	Max	86.38%	91.84%	91.98%	92.99%
	Std Dev	0.38%	0.25%	0.23%	0.25%
Noisy	**Average**	86.83%	91.98%	92.27%	93.29%
Noisy to Clean	**Difference in Average**	**-1.11%**	**-0.55%**	**-0.65%**	**-0.68%**
	Significance of Difference	0.00**	0.00**	0.00**	0.00**

The results show that keeping noise in the data consistently and significantly improves accuracy rates. This observation falsifies our second hypothesis, which stated that cleaning data prior to learning and evaluation causes an increase in POST accuracy over learning and evaluating with noisy data. Why did we observe the opposite? Looking further into the data revealed that most of the noise signals are not ambiguous. A comma, for instance, is hardly ever tagged as anything other than comma. Due to the resulting strong and unambiguous emission probability for noise symbols, we predict noise with very high accuracy.

What does that imply for modeling? Accuracy rates significantly benefit from data that contains certain entity classes that occur frequently and that are easier to predict than other categories because they are less ambiguous (not much to anyone's surprise). For boosting tagging accuracy, keeping noise in the data is beneficial. However, as true for any machine learning application, special attention needs to be paid to cross-validating a model with new data prior to making generalizations. In order to build POST models that do not overfit to the prediction of noise by overly adjusting themselves to this idiosyncrasy, noise needs to be removed from the data prior to model training. One might argue that real data are likely to contain the sort of noise that was eliminated for this project. That is true. However, not removing noise prior to learning reduced the empiric evidence that can be gathered on transitions of tags other than noise, while more information is learned about the transitions between noise tags and tags of interest. As a result, the numeric stability of transitions among relevant tags is decreased, and the predictive capability of the model for applications where correct tagging of content is favored over tagging noise is reduced.

5.3 Handling Unknowns

Applying a POS tagger to new data can result in two types of errors: misclassification of words that the model has prior knowledge about (algorithmic failure), and failure to find the right class label for a word that has not been observed by the model during training (failure in handling unknowns). Some of the newly encountered words will be correctly resolved by the algorithm by exploiting transition probabilities, while others will still be misclassified. In order to figure out if it is worthwhile to resolve unknowns after evaluating the model and prior to outputting the results we first need to understand the distribution of the two error types introduced in this section across the algorithms that we test herein.

Table 6: Error Types per Algorithm (clean data)

Error Type	UM	HMM	VitF	VitB
Unknowns	28.3%	6.1%	9.6%	1.7%
Algorithmic	71.7%	93.9%	90.4%	98.3%

Table 6 shows that for all algorithms, the vast majority of errors are due to algorithmic failures. The baseline model with about three out of ten errors being due to unknowns has by far the greatest potential for benefitting from unknown resolution. For HMM and VitF, an accuracy increase of up to 6.1% and 9.6%, respectively, is theoretically possible by associating unknowns with the right tag. For VitB we cannot expect a major accuracy improvement from unknown handling – the algorithm accomplishes most of the unknown handling by itself; the remaining errors due to unknowns might be data artifacts. This insight suggests that the more an automated solution exploits empiric evidence, the less it can be further improved by man-made post-processing strategies. For machinery that makes decisions on its own by strongly relying on its computational power and by trying to resolve uncertainties rather than admitting them, careful and well-informed engine construction is crucial since posteriori interventions cannot further improve performance. For UM, HMM and VitF, all of which exploit less empiric evidence than VitB does, a combination of an initial automated solution with hand-crafted heuristics has a potential for outperforming fully-automated approaches. Only such algorithms that are declare more uncertainties allow for posterior interventions. It is the engineer in the first place who determines how much uncertainly shall be disclosed by the engine (as for instance described in section 2, where we reason about the minimum probability for transitions and missions for the case of unknowns).

We applied the following data-driven procedure for developing post-processing rules: First, we collected all errors made by all four algorithms throughout the ten cross-fold validation on clean data. From these data we parsed out all cases in which any of the algorithms assigned "unknown" to a word after trying to solve it algorithmically. We found that neither HMM, VitF nor VitB made any mistake on unknowns that UM did not also make. Therefore we further worked only with the set of unknowns detected by UM. Next, we removed any duplicates of unknown errors (cases where unknown was assigned to the same tag-token more than once). This procedure reduced the set of unknown error by 14,105 to 29,418. We split the remaining unique unknown errors up by true tag class (true according to PTB); seeing that unknowns occurred in any but the POS and PDT class. Next, we examined the words in each class for frequent regularities per class, e.g. by analyzing patterns in endings, spelling and capitalization. Table 7 provides details on this process.

Table 7: Developing heuristics for handling unknowns

True Tag	How often classified as unknown	Rule
NNP	7429	often capitalized
NN	6390	no obvious rule, second most frequent class
JJ	5194	often dashes and/ or one out of a fixed set of endings
NNS	3522	often ends with -s
CD	1175	often contain digit(s)
VBG	1118	often ends with -ing
VBN	893	often ends with -ed
RB	832	often ends with -ly
VB	704	often ends with -e, but rule more often true for VBN, also often ends with -ize
VBD	508	often ends with -ed, but rule more often true for VBN
VBZ	475	often ends with -s, but rule more often true for NNS
NNPS	399	often capitalized and ends with -s, but rule more often true for NNP
FW	350	no obvious rule
VBP	107	no obvious rule
UH	94	no obvious rule
JJS	86	often ends with -est
JJR	56	often ends with -ier or -er
IN	42	no obvious rule
MD	21	no obvious rule
PRP	21	no obvious rule
RBR	12	often ends with -er, but rule more often true for JJR
DT	7	no obvious rule
WRB	7	often starts with wh-
PRP$	6	no obvious rule
LS	5	often contain digit(s), more often true for CD
WP	5	often starts with wh-
CC	3	no obvious rule
EX	1	no obvious rule
RBS	1	no obvious rule
RP	1	no obvious rule
SYM	1	no obvious rule
TO	1	no obvious rule
WDT	1	often starts with wh-
WP$	1	often starts with wh-

Based on the insights gained in the last step we formalized and implemented the following set of mainly orthographic rules:

1. Words containing a digit are tagged as numbers (CD).
2. Capitalized words are tagged as proper singular nouns (NNP).
3. Words ending with -ant, -able, -al, -ory, -ent, -ful, -ian, -ible, -ic, -ish, -less, -oid, or -ous are tagged as adjectives (JJ).
4. Words ending with –s are tagged as common plural nouns (NNS).

5. Words ending with -ing are tagged as present participle or gerund verbs (VBG).
6. Words ending with –ed are tagged as past participle verbs (VBN).
7. Words ending with –ly are tagged as adverbs (RB).
8. Words ending with –ize are tagged as verbs (VB).
9. Words ending with -est are tagged as adjective, superlative (JJS).
10. Words ending with -er are tagged as adjective, comparative (JJR).
11. All remaining unknowns are labeled as singular or mass noun (NN).

Next we tested the impact of these rules in the order as they are presented above on resolving unknowns. The results are shown in Tables 8 and 9. When a rule get's applied, three outcomes are possible:

- Unknowns are resolved correctly (column named *Success* in Table 8).
- Unknowns that truly belong into a different target class get assigned to the class that the rule predicts (false positives, shown in the second last column in Table 8).
- Unknowns that truly belong in the target class that the rule predicts are not resolved since the rule does not apply to them (false negatives, last column in Table 8).

After evaluating a rule (let's call this rule A) we kept rule A applied for evaluating the subsequent rule (let's call this rule B) if and only if A caused more correct tag resolutions than false positives. The following exceptions apply:

- We dropped the rule that words ending with -er are tagged as comparative adjectives. This rule correctly resolved all of the remaining seven comparative adjectives, but also converted 42 tags that belonged into other tag classes into comparative adjectives. The rule therefore overall was more damaging than helpful.
- Converting words ending with –ed caused slightly more misclassifications than correct resolutions. However, this rule reaches into the past tense verb class, and since we plan on aggregating all different verb classes into one general verb class later on we decided to keep this rule.

Finally, we examined the set of remaining false negatives per class for possible further rules. This process taught us that other rules which we identified would cause more false positives than correct resolutions, or that the generalizability of a rule per class was too low to cause a significant improvement. We assume the final set of rules to be not just corpus-specific, but of general applicability for POST.

In general, there is no standardized procedure for performing an error analysis. It requires the researcher's creativity, knowledge of the problem domain, close work with the data, and thorough analyses in order to understand the cause and nature of the errors that occur, to develop possible remedies, and to control if the application of these remedies causes negative side effects that are more harmful than the actual remedy is beneficial.

Table 8: Rule Evaluation (on clean data)

Rules				Types of errors in detecting tag			Impact of applying rule(s)			
ID	If token is unknown	Then	Other rules applied	Total	Algorith-mic	Unknown	Tokens impacted by rule	Success	Failure False Positives	Failure False Negative
1	contains digit	CD		369	62.6%	37.4%	141	126	15	12
2	capitalized	NNP	1	2217	29.1%	70.9%	2016	1561	455	10
3	ends with any of *	JJ	1,2	1550	67.9%	32.1%	546	379	167	119
4	ends with -s	NNS	1-3	678	48.4%	51.6%	431	334	97	16
5	ends with -ing	VBG	1-4	280	67.1%	32.9%	119	92	27	0
6	ends with -ed	VBN	1-5	789	89.4%	10.6%	171	83	88	1
7	ends with -ly	RB	1-5	867	93.0%	7.0%	63	60	3	1
8	ends with -ize	VB	1-5,7	1247	96.5%	3.5%	5	5	0	39
9	ends with -est	JJS	1-5,7,8	51	84.3%	15.7%	9	8	1	0
10	ends with -er	JJR	1-5,7-9	133	94.7%	5.3%	53	7	46	0
11	remainder	NN	1-5,7-9	3207	86.8%	13.2%	599	422	177	0

* -ant, -able, -al, -ory, -ent, -ful, -ian, -ible, -ic, -ish, -less, -oid, -ory, -ous

** cases in which the number of false positives exceeds correct resolutions are marked with gray background

Table 9: Rule Evaluation (on clean data)

Rules				Number of unknowns				Change in accuracy from previous rule(s)*			
ID	If token is unknown	Then	Other rules applied	UM	HMM	VitF	VitB	UM	HMM	VitF	VitB
0	and nothing else happens	error		4100	511	892	140	NA	NA	NA	NA
1	contains digit	CD		3959	508	889	140	0.117%	0.003%	0.003%	0.000%
2	capitalized	NNP	1	1943	126	193	3	1.397%	0.154%	0.401%	0.071%
3	ends with any of *	JJ	1,2	1397	123	149	3	0.351%	0.003%	0.016%	0.000%
4	ends with -s	NNS	1-3	966	93	101	3	0.310%	0.003%	0.013%	0.000%
5	ends with -ing	VBG	1-4	847	76	84	3	0.085%	0.013%	0.013%	0.000%
6	ends with -ed	VBN	1-5	676	25	29	1	0.077%	0.009%	0.014%	0.000%
7	ends with -ly	RB	1-5	613	12	16	0	0.056%	0.011%	0.011%	0.001%
8	ends with -ize	VB	1-5,7	608	12	16	0	0.005%	0.000%	0.000%	0.000%
9	ends with -est	JJS	1-5,7,8	599	12	16	0	0.007%	0.000%	0.000%	0.000%
10	ends with -er	JJR	1-5,7-9	546	9	11	0	0.006%	0.000%	0.000%	0.000%
11	remainder	NN	1-5,7-9	0	0	0	0	0.385%	0.000%	0.003%	0.000%

* cases which resulted in no accuracy gain are marked with dark gray background, cases which resulted in accuracy gains greater than zero and smaller than 0.05% are marked with light gray background

Our results show that applying our hand-crafted rules leads to statistically significant accuracy increases for all algorithms (Table 10). This confirms our third hypothesis, which assumed post-processing of unknown words to cause an increase in POST accuracy for all four algorithms. However, the rule set is capable of resolving only a small fraction of those errors that are due to unknowns (10. 5% for UM, 3.4% for HMM, 4.8% for VitF, 3.7% for VitB). To our surprise, VitF, which exploits more empiric evidence than HMM does, benefits more from a hybrid strategy (initial algorithmic solution plus rule-based post-processing) than HMM, which admits more uncertainty than VitF. VitB, the algorithm which we thought maxes out on unknown handling algorithmically, can benefit from unknown handling, but here, only 6 in 10,000 words would be impacted by this strategy. Even though the increase in accuracy due to unknown handling is smallest for VitB, this algorithm still outperforms the other three algorithms.

Table 10: Impact of Unknown Handling on Tagging Accuracy

Dataset	Measure	UM	HMM	VitF	VitB
Unknown Handling on Clean Data	**Average**	**88.68%**	**91.64%**	**92.08%**	**92.67%**
	Min	88.17%	91.28%	91.76%	92.27%
	Max	89.33%	91.99%	92.39%	93.02%
	Std Dev	0.36%	0.23%	0.20%	0.24%
Clean	**Average**	**85.72%**	**91.43%**	**91.62%**	**92.61%**
Clean to Unknown Handling	**Difference in Average**	**2.96%**	**0.21%**	**0.46%**	**0.06%**
	Significance of Difference	0.00**	0.00**	0.00**	0.00**

In summary, data-driven derivation of post-processing rules as well as rule testing are time-consuming processes that require the allocation of human resources. Our findings suggest that not investing into this strategy, but instead spending resources on building algorithms that handle uncertainties algorithmically in the first place, can lead to better performance than enhancing algorithmic solutions with hand-crafted post-processing heuristics.

5.4 Aggregation of Hidden States

The tests on tag aggregation were run on clean data and with unknown handled as described in the previous section applied. We found that consolidating the PTB tag classes (total of 36) into fewer (12), user-defined classes that are tailored to the end-user's analytical needs (see the Appendix for aggregation details) led to the highest accuracy rates accuracy across all algorithms and independent variables tested herein (Table 11, Figures 2 and 3). These results confirm our fourth hypothesis, which stated that aggregation of POST categories causes an increase in POST accuracy for all four algorithms. However, it surprised us to see that the simplest algorithm (UM) performs as well as the most complex one (VitB).

Table 11: Accuracy per algorithm and tested variable

	UM	HMM	VitF	VitB
Average	**94.26%**	**93.09%**	**94.10%**	**94.26%**
Min	93.95%	92.92%	93.93%	93.99%
Max	94.46%	93.45%	94.36%	94.60%
Std Dev	0.17%	0.16%	0.15%	0.19%

In summary, our results on aggregation suggest that an informed, needs-driven, and user-defined consolidation of available choices can lead to performance improvements that consistently across various algorithm of different complexity can have a greater positive impact than eliminating prominent error sources such as noise and unknown data. The technology that we developed for training a POS tagger can easily be reused in order to train a model with a different tag set. We emphasize the design of analytical solutions that enable end-users to interact with tools or human beings on the developmental side of solutions in such a way that customer needs can be elicited and considered for the sake of performance improvements.

Figure 2: Impact of Independent Variable on POST Accuracy

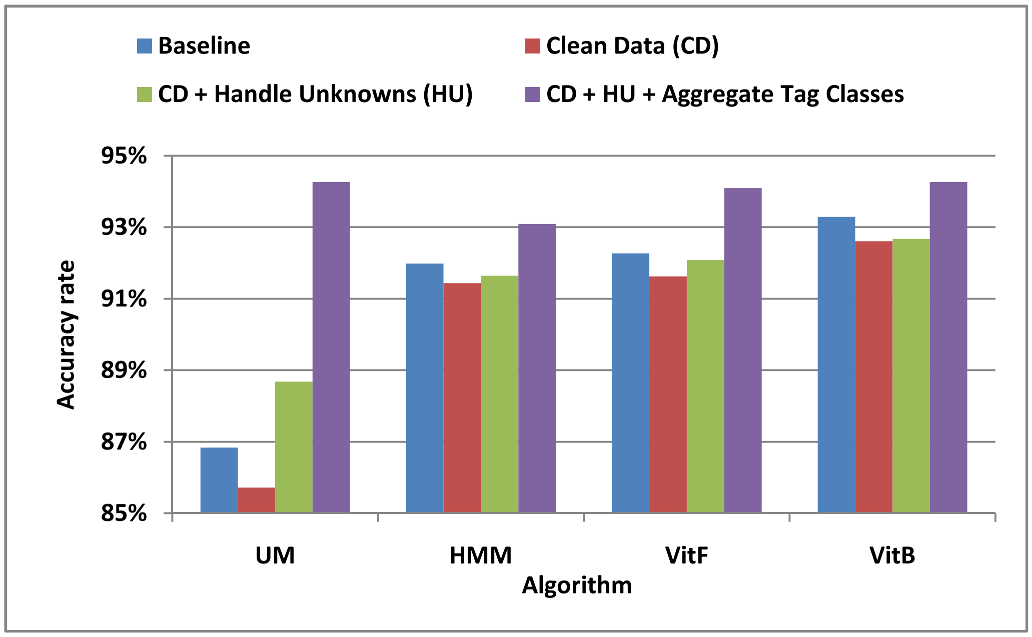
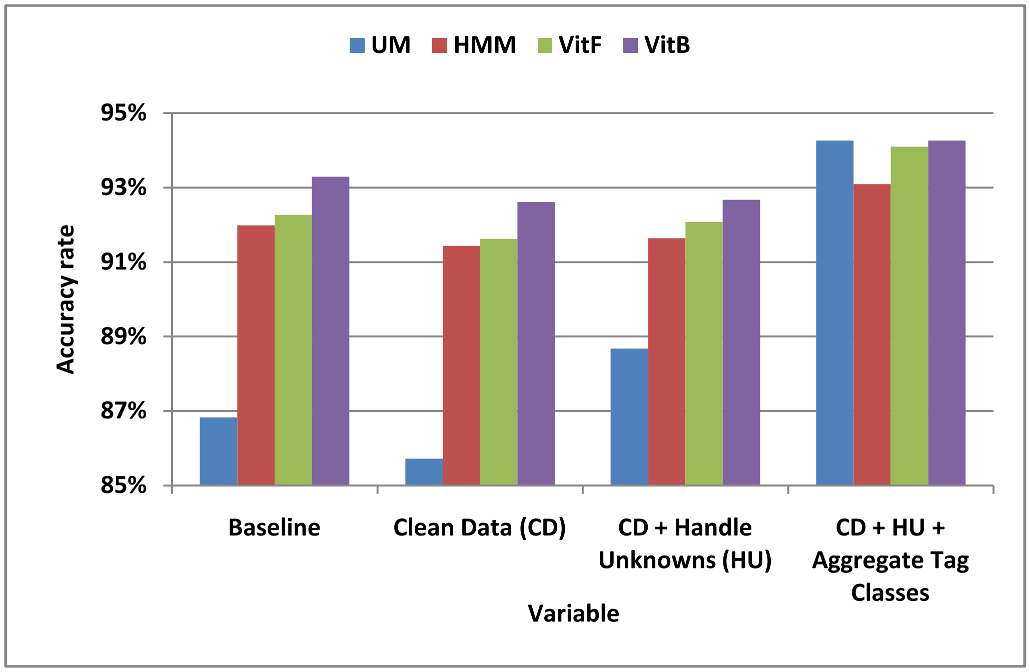


Figure 3: Impact Algorithm on POST Accuracy



6. Integration of Parts of Speech Tagging into AutoMap

Based on the insights gained from testing the impact of various independent variables on the accuracy of four different POST algorithms (the difference between the algorithms themselves being one of the variables) we decided to train the following two POS tagging models and integrate them into AutoMap:

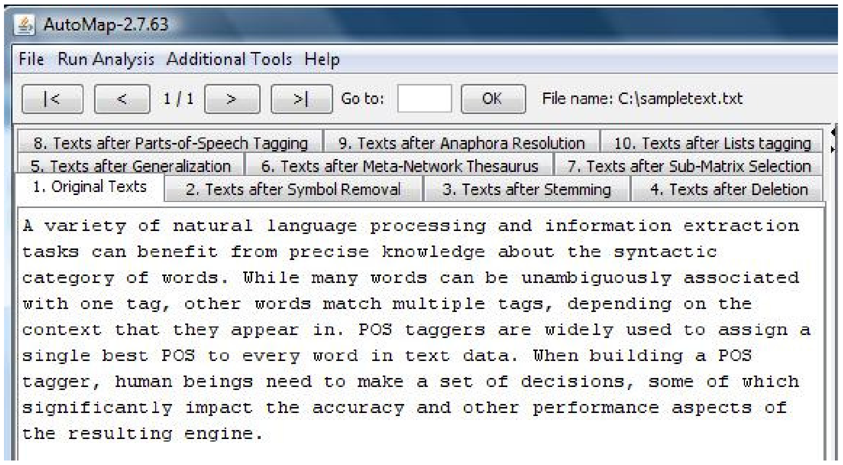
- Both models based on Viterbi with backtracing.
- One model uses the original PTB tag set, while the other model uses the aggregated tag set (Table 2).
- Each of the two models requires a separate post-processor that matches the respective tag set.

We implemented and integrated these taggers into AutoMap as follows: First, we trained both models on the full learning set (not only 90% of it), output the emission and transition matrices as data files, and added these data to AutoMap. In AutoMap, on the Utilities tab, in the Parts of Speech Tagging section, the user can chose and go back and forth between the "Tag texts using PTB tag set" option and the "Tag texts using aggregated tag set" option (Figure 5). In either case, the untagged texts will first be split into sentences by using a sentence splitter (Piao, n.d.). Next, the initialization vector will be constructed based on the tokens per sentence. Using the initialization vector as well as the states as represented in the emission and transition matrices, a trellis will be built for every sentence in the data. These trellises are used to find a complete and the most likely sequence of POS per words per sentence. Users can use the POS tagger in the GUI or batch mode version of AutoMap in two ways (Carley, Diesner, Reminga, & Tsvetovat, 2007):

- Stand-alone feature: When either "Tag Texts…" option is selected, AutoMap performs POST and displays each word along with the POS that the tagger predicted for it. The user can store the POS annotated corpus. For the sample text shown in Figure 4, AutoMap generated the POS annotated text shown in Figures 5 and 6.
- Output a table (coma separated values format) that lists all words in a corpus in the first column and the respective POS that the model has identified for that word in the following column. If more than one POS was predicted for a word, the word-tag tuples will be placed in multiple rows. Tables 12 and 13 show that list for the sample text given in Figure 4 using the tagger trained on the full PTB tag set.

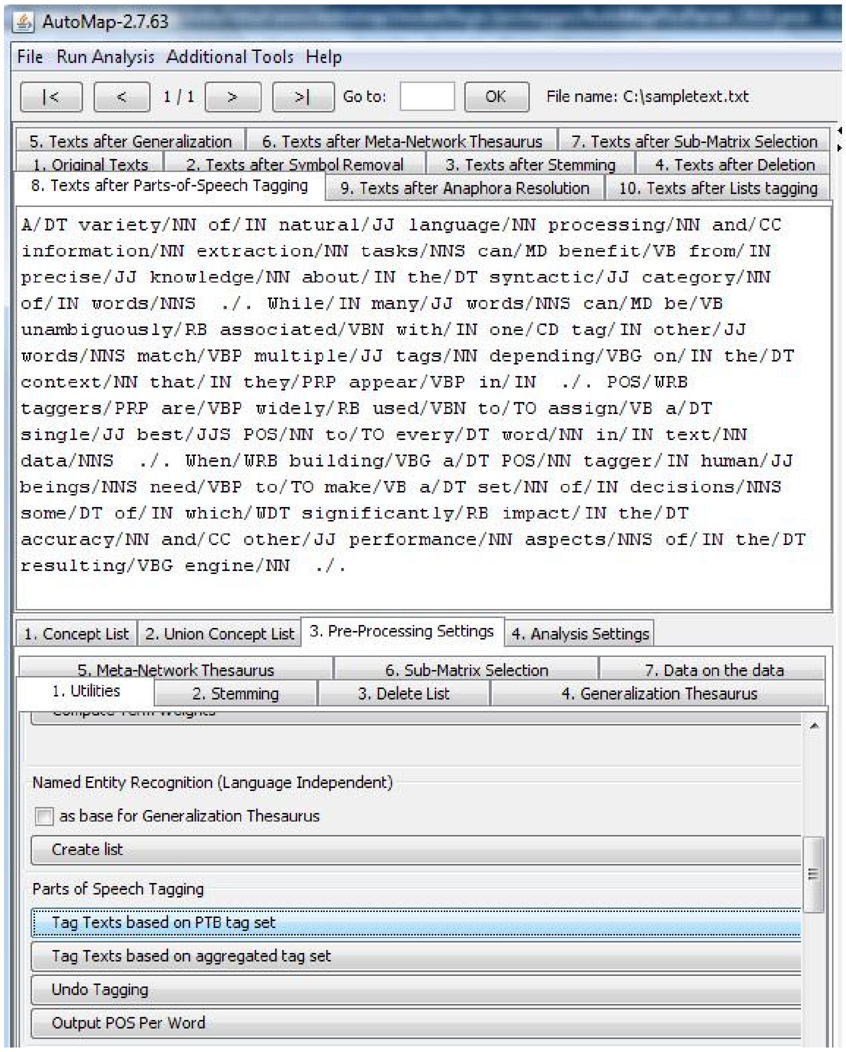
Besides supporting a variety of NLP and IE routines, AutoMap's main purpose is to facilitate content analysis as well as the extraction of one- and multi-mode networks from texts (Diesner & Carley, 2004, 2006; McConville, et al., 2008). When relational data is extracted with AutoMap, outputs can be stored as DyNetML files (DyNetML is an XML derivate designed for graph representation (Carley, et al., 2007). DyNetML files represent one or multiple graphs that comprise vertices and edges. The nodes and edges can hold attributes. POS are one possible node attribute. ORA (Carley, et al., 2007), a software for relational data analysis, can read DyNetML files and run several reports that consider POS in the computation of network analytic measures.

Figure 4: Raw text loaded into AutoMap


AutoMap-2.7.63
File Run Analysis Additional Tools Help
|< < 1 / 1 > >| Go to: OK File name: C:\sampletext.txt
8. Texts after Parts-of-Speech Tagging
9. Texts after Anaphora Resolution
10. Texts after Lists tagging
5. Texts after Generalization
6. Texts after Meta-Network Thesaurus
7. Texts after Sub-Matrix Selection
1. Original Texts
2. Texts after Symbol Removal
3. Texts after Stemming
4. Texts after Deletion

```
A variety of natural language processing and information extraction
tasks can benefit from precise knowledge about the syntactic
category of words. While many words can be unambiguously associated
with one tag, other words match multiple tags, depending on the
context that they appear in. POS taggers are widely used to assign a
single best POS to every word in text data. When building a POS
tagger, human beings need to make a set of decisions, some of which
significantly impact the accuracy and other performance aspects of
the resulting engine.
```

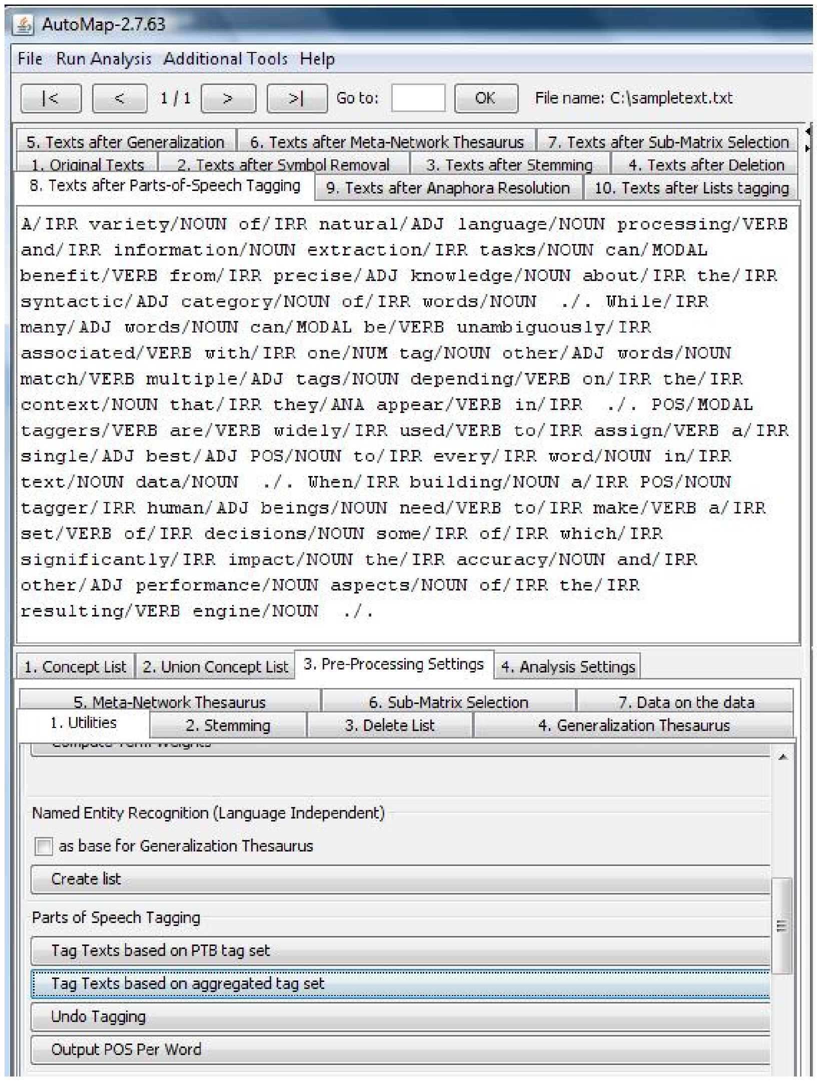
Figure 5: Integration of POS Tagger based on PTB tag set into AutoMap as stand-alone feature


AutoMap-2.7.63
File Run Analysis Additional Tools Help
|< < 1 / 1 > >| Go to: OK File name: C:\sampletext.txt
5. Texts after Generalization
6. Texts after Meta-Network Thesaurus
7. Texts after Sub-Matrix Selection
1. Original Texts
2. Texts after Symbol Removal
3. Texts after Stemming
4. Texts after Deletion
8. Texts after Parts-of-Speech Tagging
9. Texts after Anaphora Resolution
10. Texts after Lists tagging

```
A/DT variety/NN of/IN natural/JJ language/NN processing/NN and/CC
information/NN extraction/NN tasks/NNS can/MD benefit/VB from/IN
precise/JJ knowledge/NN about/IN the/DT syntactic/JJ category/NN
of/IN words/NNS  ./. While/IN many/JJ words/NNS can/MD be/VB
unambiguously/RB associated/VBN with/IN one/CD tag/IN other/JJ
words/NNS match/VBP multiple/JJ tags/NN depending/VBG on/IN the/DT
context/NN that/IN they/PRP appear/VBP in/IN  ./. POS/WRB
taggers/PRP are/VBP widely/RB used/VBN to/TO assign/VB a/DT
single/JJ best/JJS POS/NN to/TO every/DT word/NN in/IN text/NN
data/NNS  ./. When/WRB building/VBG a/DT POS/NN tagger/IN human/JJ
beings/NNS need/VBP to/TO make/VB a/DT set/NN of/IN decisions/NNS
some/DT of/IN which/WDT significantly/RB impact/IN the/DT
accuracy/NN and/CC other/JJ performance/NN aspects/NNS of/IN the/DT
resulting/VBG engine/NN  ./.
```

1. Concept List
2. Union Concept List
3. Pre-Processing Settings
4. Analysis Settings
5. Meta-Network Thesaurus
6. Sub-Matrix Selection
7. Data on the data
1. Utilities
2. Stemming
3. Delete List
4. Generalization Thesaurus
Named Entity Recognition (Language Independent)
as base for Generalization Thesaurus
Create list
Parts of Speech Tagging
Tag Texts based on PTB tag set
Tag Texts based on aggregated tag set
Undo Tagging
Output POS Per Word

Figure 6: Integration of POS Tagger based on aggregated tag set into AutoMap as stand-alone


AutoMap-2.7.63
File Run Analysis Additional Tools Help
|< < 1 / 1 > >| Go to: OK File name: C:\sampletext.txt
5. Texts after Generalization
6. Texts after Meta-Network Thesaurus
7. Texts after Sub-Matrix Selection
1. Original Texts
2. Texts after Symbol Removal
3. Texts after Stemming
4. Texts after Deletion
8. Texts after Parts-of-Speech Tagging
9. Texts after Anaphora Resolution
10. Texts after Lists tagging
1. Concept List
2. Union Concept List
3. Pre-Processing Settings
4. Analysis Settings
5. Meta-Network Thesaurus
6. Sub-Matrix Selection
7. Data on the data
1. Utilities
2. Stemming
3. Delete List
4. Generalization Thesaurus
Named Entity Recognition (Language Independent)
as base for Generalization Thesaurus
Create list
Parts of Speech Tagging
Tag Texts based on PTB tag set
Tag Texts based on aggregated tag set
Undo Tagging
Output POS Per Word

```
A/IRR variety/NOUN of/IRR natural/ADJ language/NOUN processing/VERB
and/IRR information/NOUN extraction/IRR tasks/NOUN can/MODAL
benefit/VERB from/IRR precise/ADJ knowledge/NOUN about/IRR the/IRR
syntactic/ADJ category/NOUN of/IRR words/NOUN  ./. While/IRR
many/ADJ words/NOUN can/MODAL be/VERB unambiguously/IRR
associated/VERB with/IRR one/NUM tag/NOUN other/ADJ words/NOUN
match/VERB multiple/ADJ tags/NOUN depending/VERB on/IRR the/IRR
context/NOUN that/IRR they/ANA appear/VERB in/IRR  ./. POS/MODAL
taggers/VERB are/VERB widely/IRR used/VERB to/IRR assign/VERB a/IRR
single/ADJ best/ADJ POS/NOUN to/IRR every/IRR word/NOUN in/IRR
text/NOUN data/NOUN  ./. When/IRR building/NOUN a/IRR POS/NOUN
tagger/IRR human/ADJ beings/NOUN need/VERB to/IRR make/VERB a/IRR
set/VERB of/IRR decisions/NOUN some/IRR of/IRR which/IRR
significantly/IRR impact/NOUN the/IRR accuracy/NOUN and/IRR
other/ADJ performance/NOUN aspects/NOUN of/IRR the/IRR
resulting/VERB engine/NOUN  ./.
```

Internally, AutoMap uses POST as one out of multiple decision support features for:

1. Named Entities Extraction, which identifies relevant types of information that are referred to by a name, such as people, organizations, and locations.
2. Anaphora Resolution, which converts personal pronouns into the actual social entities that those pronouns refer to.

How can end users exploit POST for text analysis projects? We envision a variety of potential usages:

1. Data reduction in the sense of deleting non-content bearing words from texts: Though it ultimately depends on the user and application domain what the set of "non-content" words entails, such concepts often belong to one of categories that we aggregated in the IRR class. Users can output the word-POS tuple table and add the words that are classified as IRR to a delete list. When applying a delete list, AutoMap searches the

Table 12: POS per Word (part1)

Word	Tag	Frequency
.	.	4
a	irr	4
about	irr	1
accuracy	noun	1
and	irr	2
appear	verb	1
are	verb	1
aspects	noun	1
assign	verb	1
associated	verb	1
be	verb	1
beings	noun	1
benefit	verb	1
best	adj	1
building	noun	1
can	modal	2
category	noun	1
context	noun	1
data	noun	1
decisions	noun	1
depending	verb	1
engine	noun	1
every	irr	1
extraction	irr	1
from	irr	1
human	adj	1
impact	noun	1
in	irr	2
information	noun	1
knowledge	noun	1
language	noun	1
make	verb	1
many	adj	1
match	verb	1
multiple	adj	1
natural	adj	1

Table 13: POS per Word (part2)

Word	Tag	Frequency
needs	verb	1
of	irr	5
on	irr	1
one	num	1
other	adj	2
performance	noun	1
pos	modal	1
precise	adj	1
processing	verb	1
resulting	verb	1
set	verb	1
significantly	irr	1
single	adj	1
some	irr	1
syntactic	adj	1
tag	noun	1
tagger	irr	1
taggers	verb	1
tags	noun	1
tasks	noun	1
text	noun	1
that	irr	1
the	irr	4
they	ana	1
to	irr	3
unambiguously	irr	1
used	verb	1
variety	noun	1
when	irr	1
which	irr	1
while	irr	1
widely	irr	1
with	irr	1
word	noun	1
words	noun	3

texts that are currently loaded for the words specified in the delete list and removes any matches by either dropping them completely or inserting a placeholder at the position where a word was removed (this choice is made by the user). In order to remove noise that does not occur in word form, words being associated with the SYM can also be added to the delete list.

2. Named Entity Extraction: the AGENTLOC class collects instances of individual agents and locations from the user's data, and the ORG class comprises instances of organizations or other mentions of multiple people. Retrieving these entities and performing network text analysis on them in AutoMap can help people to explore the social and spatial network(s) represented in their data. Since POST operates on a word-by-word basis, identifying agents, organizations and locations that occur as N-

grams (e.g. *Henry Ford* or *Occupational Safety and Health Administration*) implies searching the POS annotated corpus for collocations of the AGENTLOC or ORG tag.

3. Identification of social structure: One application of AutoMap is the approximation of relational data that is represented in text data. AutoMap supports the extraction of two types of relational data: one-mode networks (all nodes are of the same type) and multi-mode networks (nodes can be associated with different node classes). By default, all nodes in a one-mode network belong to the node class *knowledge*, while in multi-mode networks, nodes can belong to one or multiple of the classes *agent*, *organization*, *task/event*, *resource*, *knowledge*, *location*, and *time*. Revealing and further analyzing relational data helps people in going beyond the identification of social networks and to also answer questions like: Who is located where, and what people or groups have access to what resources, tasks, and knowledge? Further analysis of multi-mode relational data (multiple node classes, such as agent and action) has helped people to understand the benefits or risks that a certain network structure implies for a socio-technical system (Carley, et al., 2007). For such projects, the words in the VERB class could serve as events or tasks, nouns could be screened for resources, and the MODAL class might serve as node or edge attributes. Instances of various node classes found this way could be further cross-verified or supplemented by using other techniques that support users in automatically finding instances of user-defined ontology classes in texts (Bikel, et al., 1999; Diesner & Carley, 2008).
4. Identification of node attributes: One-mode network extraction has been used to reveal mental models of (groups of) people. Mental models are considered to represent the reality that people have in their minds and use to make sense of their surroundings, or the cognitive constructs that reflect people's knowledge and information about a certain topic. Multi-mode network extraction serves the exploration of network configuration as described under the previous point. People are not bound to those categories, but can use their own ontologies or taxonomies in AutoMap (Diesner & Carley, 2008). Whether using the default or self-defined node classification schemata, and whether extracting one- or multi-mode networks, people can also extract attributes on nodes. The ADJ class might be an appropriate candidate for providing suggestions for words that qualify as node attributes.

7. Limitations and Conclusions

Several limitations apply to the work presented herein. First, even though the training and testing set (PTB corpus) contains more than a million data points, it still reflects a certain time period, style (journalistic writing) and range of domains (news paper articles). Applying the constructed POST models to data that differs in any of these dimensions is likely to result in accuracy rates lower than the ones reported herein. Second, we did not test MM of a higher order. For data sets with lengthy sentences, e.g. academic writing, or for data in that N-grams of size larger than size two are crucial and occur often, using a MM of a higher order might further improve tagging accuracy while also increasing computational complexity. Finally, all algorithms tested are stochastic taggers; thus that a comparison to accuracy rates achieved with rule- or transformation-based systems could be valuable.

The POS taggers that we implemented into AutoMap performed reasonably well on tagging texts that were unseen during training the models. What does reasonably well mean? Overall, our accuracy rates are a few (about three to four) percent lower than the best accuracy rates (96% to 97%) published for POS taggers that were built using PTB (Jurafsky & Martin, 2000). Let us look at our accuracy rates in more detail: If the first of our tagger in AutoMap (trained on clean data, performing unknown handling, using full PTB tag set) was used to tag a 20 word sentence, it would mislabeled two to three (precisely 2.3) words when using UM, and one to two words when using HMM (1.7), VitF (1.6) or VitB (1.5). If we the second tagger (trained on clean data, performing unknown handling, using aggregated tag set), it would mislabel about one word in a 20-word sentence (1.1 for UM, 1.4 for HMM, 1.2 for VitF and VitB). Using this second tagger, the probability that all words in 20 word sentence would get tagged correctly is 31% for UM and VitB, 24% for HMM, and 29% for VitF.

Besides adding a well-performing POS tagger to AutoMap, our goal with this project was to look under the hood of MM-based, stochastic POST in order to understand how certain variables impact the resulting POST accuracy. The main contribution of this report is to quantify and reason about the change in tagging accuracy that is due to choices about design decisions that human beings need to make when implementing a stochastic POS tagger. Table 12 shows our hypotheses and respective findings (** indicate significance for a confidence interval of 95%). The remainder of the report summarizes our lessons learned.

Table 12: Summary of results of hypothesis testing

Hypothesis	UM	HMM	VitF	VitB
H1: POST accuracy increases from step to step, so that: - accuracy with HMM is higher than with UM - accuracy with VitF is higher than with HMM - accuracy with VitB is higher than with VitF.	N.A.	Yes**	Yes**	Yes**
H2: Data cleaning prior to learning and evaluation causes an increase in POST accuracy over learning and evaluating with noisy data for all for algorithms.	No**	No**	No**	No**
H3: Post-processing of unknown words causes an increase in POST accuracy for all four algorithms.	Yes**	Yes**	Yes**	Yes**
H4: Aggregation of POST categories causes an increase in POST accuracy for all four algorithms.	Yes**	Yes**	Yes**	Yes**

We have shown how design decisions about computational solutions for common NLP tasks, here POST, can significantly impact the behavior of the resulting engine. The empirical comparison of four POS algorithms, which all are integral parts of the Viterbi algorithm, confirmed our assumption that an increase in the empirical evidence that an algorithm identifies and exploits causes increases in accuracy rates. Therefore, the upgrade from local search to global search leads to improvements in accuracy at the expense of higher computational complexity. This investment pays off most if the search space is traversed through for the best solution not only in a forward fashion, but with a bidirectional search.

Removing noise from the training data prior to learning a model leads to significant decreases in accuracy rates while the amount and numerical stability of the learned probabilities for the tags of interest increase. We argue that the generalizability of the model benefits from the decision to remove noise.

Across all algorithms tested, the majority of errors were due to algorithmic failures, while only a small portion of errors was caused by labeling newly encountered words after trying to resolve them algorithmically as unknowns. We showed that when building POS taggers, one can lower the ratio of unknown handling errors by developing and adding post-processing rules for handling new words. However, the process of constructing and testing unknown handling rules is fairly labor- and time intense, and can be avoided by designing algorithms that exploit as much empiric evidence as possible to begin with. We learned that the more an algorithm is designed towards admitting uncertainties rather than trying to resolve them algorithmically on its own, the more hybrid strategies of initial algorithmic solutions plus manually constructed post-processing heuristics can improve accuracy.

Across all independent variables tested in this project we observed the strongest performance improvement when the tag set was aggregated and reduced to fewer categories that are tailored towards the user's needs. We therefore advocate the development of models and tools that allow end-users to specify or participate in the consolidation of categories out of a predefined pool of choices according to their requirements.

We conclude that error rates reported on POS taggers and obtained by users who work with such tools highly depend on choices about design decisions that have to be made when building a tagger. Therefore, the variables that significantly impact a tagger's performance need to be identified and their effect on the tagger needs to be measured and reported so that everyone - developers and users - can learn about the sensitivity of the engine and responsibly work with such systems.

References

Arguello, J., & Rose, C. P. (2006). Museli: A Multi-source Evidence Integration Approach to Topic Segmentation of Spontaneous Dialogue. *Proceedings of the North American Chapter of the Association for Computational Linguistics (short paper)*.

Atwell, E. (1987). Constituent-likelihood grammar. In R. Garside, G. Sampson & G. Leech (Eds.), *The computational analysis of English: a corpus-based approach*. London: Longman.

Baum, L. (1972). An inequality and associated maximization technique in statistical estimation for probabilistic functions of a Markov process. *Inequalities, 3*, 1-8.

Bikel, D., M. , Schwartz, R., & Weischedel, R., M. (1999). An Algorithm that Learns What's in a Name, *Machine Learning* (Vol. 34, pp. 211-231): Kluwer Academic Publishers.

Carley, K. M., Diesner, J., Reminga, J., & Tsvetovat, M. (2007). Toward an interoperable dynamic network analysis toolkit. *Decision Support Systems. Special Issue Cyberinfrastructure for Homeland Security, 43*(4), 1324-1347.

Church, K. (1988). *A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text.* Paper presented at the 2nd Conference on Applied Natural Language Processing, Austin, TX.

DeRose, S. (1988). Grammatical category disambiguation by statistical optimization. *Computational Linguistics, 14*, 31-39.

Diesner, J., & Carley, K. M. (2004). *AutoMap1.2 - Extract, analyze, represent, and compare mental models from texts*: Carnegie Mellon University, School of Computer Science, Institute for Software Research International, Technical Report.

Diesner, J., & Carley, K. M. (2006). Revealing Social Structure from Texts: Meta-Matrix Text Analysis as a novel method for Network Text Analysis. In V. K. Narayanan & D. J. Armstrong (Eds.), *Causal Mapping for Information Systems and Technology Research: Approaches, Advances, and Illustrations* (pp. 81-108). Harrisburg, PA: Idea Group Publishing.

Diesner, J., & Carley, K. M. (2008). Conditional Random Fields for Entity Extraction and Ontological Text Coding. *Journal of Computational and Mathematical Organization Theory, 14*, 248 - 262.

Dietterich, T. G. (2002). *Machine Learning for Sequential Data: A Review.* Paper presented at the Joint IAPR International Workshops SSPR 2002 and SPR 2002, August 6-9, 2002, Windsor, Ontario, Canada.

Jurafsky, D., & Martin, J. H. (2000). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*: Prentice Hall PTR.

Klein, D., & Manning, C. D. (2002). *Conditional structure versus conditional Estimation in NLP models.* Paper presented at the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP-02), Philadelphia, USA.

Krovetz, B. (1995). *Word sense disambiguation for large text databases.* University of Massachusetts, Amherst.

Kupiec, J. (1992). Robust part-of-speech tagging using a hidden Markov model. *Computer Speech and Language, 6*, 225-242.

Lappin, S., & Leass, H. J. (1994). An algorithm for pronominal anaphora resolution. *Comput. Linguist., 20*(4), 535-561.

Manning, C., & Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. Cambridge, MA: MIT Press.

McConville, E., Diesner, J., & Carley, K. M. (2008). *Software demonstration of AutoMap*. Paper presented at the XXVIII Sunbelt Social Network Conference.

Mitchell, P. M., Santorini, B., & Marcinkiewicz, M. A. (1993). Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics, 19*(2), 313-330.

Piao, S. S. (n.d.). English sentence breaker. http://text0.mib.man.ac.uk:8080/scottpiao/sent_detector.

Porter, M. F. (1980). An algorithm for suffix stripping. *Program, 14*(3), 130−137.

Rabiner, L. R. (1989). A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proc. IEEE 77, 2*, 257-285.

Stolz, W. S., Tannenbaum, P. H., & Carstensen, F. V. (1965). Stochastic Approach to the Grammatical Coding of English. *Communications of the ACM, 8*, 399–405.

Viterbi, A. J. (1967). Error bounds for convolutional codes and asymptotically optimal decoding algorithm. *IEEE Transactions on Information Theory, 13*, 260-269.

Weischedel, R., Meter, M., Schwartz, R., Ramshaw, L., & Palmucci, J. (1993). Coping with ambiguity and unknown words through probabilistic models. *Computational Linguistics, 19*(2), 359-382.

Appendix: PTB Tagset

PTB Tag	Meaning	Aggregated Tag	Instances in PTB
NN	noun, common, singular or mass	NOUN	161397
IN	preposition or conjunction, subordinating	IRR	136714
DT	determiner	IRR	116454
JJ	adjective or numeral, ordinal	ADJ	76586
NNP	noun, proper, singular	AGENT	62020
NNS	noun, common, plural	NOUN	55912
RB	adverb	IRR	52037
PRP	pronoun, personal	ANA	47303
VBD	verb, past tense	VERB	46684
CC	conjunction, coordinating	IRR	38097
VB	verb, base form	VERB	36887
VBN	verb, past participle	VERB	29435
TO	to as preposition or infinitive marker	IRR	26135
VBZ	verb, present tense, 3rd person singular	VERB	21627
VBG	verb, present participle or gerund	VERB	17255
PRP$	pronoun, possessive	IRR	16918
CD	numeral, cardinal	NUM	15178
VBP	verb, present tense, not 3rd person singular	VERB	14371
MD	modal auxiliary	MODAL	14115
:	:	SYM	10917
''	''	SYM	9201
``	``	SYM	8838
POS	genitive marker	POS	5247
WDT	WH-determiner	IRR	4990
WP	WH-pronoun	IRR	4732
WRB	Wh-adverb	IRR	4625
JJR	adjective, comparative	ADJ	2914
)	)	SYM	2506
(	(	SYM	2477
EX	existential there	IRR	2224
NNPS	noun, proper, plural	ORG	1958
RBR	adverb, comparative	IRR	1901
JJS	adjective, superlative	ADJ	1743
RP	particle	IRR	1630
SYM	symbol	SYM	1268
UH	interjection	IRR	883
FW	foreign word	FW	803
RBS	adverb, superlative	IRR	784
PDT	pre-determiner	IRR	728
$	$	SYM	579
LS	list item marker	SYM	446
WP$	WH-pronoun, possessive	IRR	251